Exhibit B

THE HONORABLE JAMES L. ROBART

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WASHINGTON
SEATTLE DIVISION

CYWEE GROUP LTD., ***Plaintiff,*** **v.** **HTC CORPORATION and HTC AMERICA, INC.,** ***Defendants.***	**Civil Action No. 2:17-cv-00932-JLR** **PLAINTIFF'S DISCLOSURE OF ASSERTED CLAIMS AND INFRINGMENT CONTENTIONS**

Pursuant to Patent Local Rule 120 and the Court's Minute Order Setting Trial Dates and Related Dates (Dkt. No. 42), Plaintiff CyWee Group Ltd. ("CyWee") serves its Disclosure of Asserted Claims and Infringement Contentions regarding U.S. Patent Nos. 8,441,438 (the "'438 patent") and 8,552,978 (the "'978 patent").

A. Each claim ("Asserted Claim") of each patent in suit that is allegedly infringed by each opposing party, including for each claim the applicable statutory subsections of 35 U.S.C. § 217 asserted.

Patent	Infringed Claims	Statutory Subsections
'438 patent	1, 3, 4, 5, 14, 15, 16, 17, 19	35 U.S.C. §§ 271(a)-(b)

SHORE CHAN DEPUMPO LLP
901 MAIN ST., STE. 3300
DALLAS, TEXAS 75202
(T) 214.593.9110

Patent	Infringed Claims	Statutory Subsections
’978 patent	10, 12	35 U.S.C. §§ 271(a)-(b)

B. For each Asserted Claim, each accused apparatus, product, device, process, method, act, or other instrumentality (“Accused Device”) of each opposing party. Each product, device, and apparatus must be identified by name or model number, if known. Each method or process must be identified by name, if known, or by any product, device, or apparatus which, when used, allegedly results in the practice of the claimed method or process.

Each of the following Accused Devices infringes each asserted claim of the ’438 patent: the HTC One M9, HTC One A9, HTC 10, HTC Bolt, HTC U Ultra, HTC U11, and HTC U11 Life.

Each of the following Accused Devices infringes each asserted claim of the ’978 patent: the HTC One M9, HTC One A9, HTC 10, HTC Bolt, HTC U Ultra, HTC U11, and HTC U11 Life.

C. A chart identifying specifically where each element of each Asserted Claim is found within each Accused Device, including for each claim element that such party contends is governed by 35 U.S.C. § 112(6), the identity of the structure(s), act(s), or material(s) in the Accused Device that performs the claimed function.

Claim charts are provided as Exhibits 1 through 14. Each claim chart is an exemplar of how all HTC devices manufactured using the same or similar technology infringe each asserted claim. Nothing in these claim charts is intended to prevent CyWee from presenting additional evidence of infringement at trial.

D. For each claim which is alleged to have been indirectly infringed, an identification of any direct infringement and a description of the acts of the alleged indirect infringer that contribute to or are inducing that direct infringement. Insofar as alleged direct infringement is based on joint acts of multiple parties, the role of each such party in the direct infringement must be described.

CyWee contends that HTC directly infringes each Asserted Claim of the patents-in-suit.

Further, at least as a result of the filing of the complaints in this case, HTC is aware of the patents-in-suit, is aware that its actions with regards to distributors, resellers, and/or end users of the accused products would induce infringement, and despite such awareness takes active steps, such as dissemination of the Accused Devices, and product manuals, instructions, promotional and marketing materials, and/or technical materials to distributors, resellers, and end users, encouraging infringement of the patents-in-suit. HTC's infringing actions further include HTC's release, in the United States, of the HTC U11 (in or around December 2017) and HTC U11 Life (in or around November 2017) after CyWee filed its complaints in this case, as well as its continued sales of other infringing products.

HTC sells its products to distributors and resellers with the expectation that they will sell said devices to end users. Distributors and resellers of HTC products, including AliExpress, Amazon, Best Buy, eGlobal Central, Groupon, HTC, Jet.com, Newegg, Sprint, TecoBuy, T-Mobile, Verizon, Walmart, and World Wide Voltage each sell Accused Devices, and thereby directly infringe the patents-in-suit.

	HTC One M9	HTC One A9	HTC 10	HTC Bolt	HTC U Ultra	HTC U11	HTC U11 Life
AliExpress		X	X			X	
Amazon	X		X	X	X	X	
Best Buy	X	X	X	X		X	
eGlobal Central			X		X	X	
Groupon		X					
HTC	X	X	X	X	X	X	X
Jet.com		X	X				
Newegg	X		X				
Sprint		X	X	X		X	X
TecoBuy			X		X	X	
T-Mobile	X		X				X
Verizon	X		X				
Walmart	X	X	X	X	X	X	

	HTC One M9	HTC One A9	HTC 10	HTC Bolt	HTC U Ultra	HTC U11	HTC U11 Life
World Wide Voltage	X	X				X	

Distributors and resellers of Accused Devices demonstrate usage of said devices to end users, which is itself an infringing act, and teaches said end users how to infringe CyWee's patents. Distributors and resellers of the Accused Devices further disseminate product manuals, promotional materials, and/or technical materials to end users.

End users of the patents-in-suit directly infringe through normal and ordinary use of the Accused Devices as described in CyWee's Opposition to HTC's Motion to Dismiss (Dkt. No. 39) and the Declaration of Dr. Nicholas Gans (Dkt. No. 20-3), both of which are incorporated by reference herein. CyWee attached detailed claim charts to both its original complaint (Dkt. No. 1) and first amended complaint (Dkt. No. 20) showing how those products infringe when used by persons such as and including end users. Further, the charts attached hereto show how the Accused Devices infringe when used. Those charts show that HTC touts inclusion of an accelerometer, gyroscope, and magnetometer. *See* https://www.htc.com/us/smartphones/htc-bolt/buy/. As CyWee's expert testified in a declaration attached to CyWee's amended complaint, the patented inventions teach how to determine a device's current orientation based on motion data detected by its motion sensors, such as an accelerometer, gyroscope, and magnetometer. Dkt. No. 20-3 ¶ 8.

HTC encourages the use of advanced motion sensor features, and CyWee expects that discovery will confirm that these features depend on its claimed technology. For example, HTC advertises its "Motion Launch" and "Motion gestures" features on its website, and provides instructions for using said features on its website. Those instructions state that a user can answer an HTC phone by picking it up and raising it to his or her head, and that the phone can be muted by placing it facedown. *E.g.*, http://www.htc.com/us/support/htc-u11-sprint/howto/motion-

gestures.html. Further examples follow: http://www.htc.com/us/support/htc-bolt/howto/what-is-motion-launch.html (Motion Launch instructions for HTC Bolt); http://www.htc.com/us/support/htc-u-ultra/howto/what-is-motion-launch.html (Motion Launch instructions for HTC U Ultra); http://www.htc.com/us/support/htc-one-m9-att/howto/what-is-motion-launch.html (Motion Launch instructions for HTC One M9 sold by AT&T); http://www.htc.com/us/support/htc-10/howto/what-is-motion-launch.html (Motion Launch instructions for HTC 10); http://www.htc.com/us/support/htc-one-m9-t-mobile/howto/616527.html (Motion Launch instructions for HTC One M9 sold by T-Mobile); http://www.htc.com/us/support/htc-one-m9/howto/603803.html (Motion Launch instructions for HTC One M9); http://www.htc.com/us/support/htc-one-m9-sprint/howto/what-is-motion-launch.html (Motion Launch instructions for HTC One M9 sold by Sprint); http://www.htc.com/us/support/htc-bolt/howto/motion-gestures.html (showing usage of motion gestures for HTC Bolt); http://www.htc.com/us/support/htc-one-m9-sprint/howto/motion-gestures.html (showing usage of motion gestures for HTC One M9); http://www.htc.com/us/support/htc-10/howto/motion-gestures.html (showing usage of motion gestures for HTC 10); http://www.htc.com/us/support/htc-one-acgc-spire/howto/389432.html (showing usage of motion gestures for HTC One); http://www.htc.com/us/support/htc-u-ultra/howto/motion-gestures.html (showing usage of motion gestures for HTC U Ultra); http://www.htc.com/us/support/htc-one-a9/howto/686277.html (showing usage of motion sensors for HTC One A9); http://www.htc.com/us/support/htc-u11/howto/motion-gestures.html (showing usage of motion gestures for HTC U11); http://www.htc.com/us/support/htc-u11-sprint/howto/motion-gestures.html (showing usage of motion gestures for HTC U11sold by Sprint). HTC distributors, such as Sprint, propagate and distribute instructions for using these features. *See* https://support.sprint.com/support/tutorial/Set-up-motion-launch-gestures-HTC-Onereg-M9/WScenario_542_59211_771_en_1997-dvc8870002prd. Many of the features described (such as answering a phone by picking it up or muting it by placing the phone face

down) require motion sensors, and, as described in CyWee's charts attached hereto, Android code running on those devices fuses data from sensors on HTC's phones in an infringing manner.

This case is in its infancy, and CyWee expects to receive information related to HTC's induced infringement through discovery. Accordingly, CyWee explicitly reserves the right to seek leave to amend its infringement contentions as the case progresses.

E. Whether each element of each asserted claim is claimed to be literally present or present under the doctrine of equivalents in the Accused Device.

CyWee contends that each asserted claim is literally infringed by HTC's accused products, as indicated by the claim charts referenced above.

F. For any patent that claims priority to an earlier application, the priority date to which each asserted claim allegedly is entitled.

CyWee alleges that all asserted claims of the '438 patent and claims 10 and 12 of the '978 patent are entitled to a priority date as of January 6, 2010, based on the date of provisional application no. 61/292,558. But CyWee further alleges that those claims are entitled to an earlier priority date based on work related to the JIL Phone prototype. More specifically, CyWee alleges that all asserted claims of the '438 patent are entitled to a priority date of July 29, 2009. CyWee further alleges that claims 10 and 12 of the '978 patent are entitled to a priority date of September 25, 2009.

Dated: December 29, 2017

Respectfully submitted,

/s/William D. Ellerman

Carmen E. Bremer, WSBA 47,565
carmen.bremer@bremerlawgroup.com
BREMER LAW GROUP PLLC
1700 Seventh Avenue, Suite 2100
Seattle, WA 98101
T: (206) 357-8442
F: (206) 858-9730

David A. Lowe, WSBA 24,453
Lowe@LoweGrahamJones.com
Tim J. Billick, WSBA No. 46,690
Billick@LoweGrahamJones.com
LOWE GRAHAM JONESPLLC
701 Fifth Avenue, Suite 4800
Seattle, WA 98104
T: 206.381.3300
F: 206.381.3301

Michael W. Shore*
Alfonso G. Chan*
Christopher Evans*
Ari B. Rafilson*
William D. Ellerman*
Paul T. Beeler*
SHORE CHAN DEPUMPO LLP
901 Main Street, Suite 3300
Dallas, Texas 75202
Telephone (214) 593-9110
Facsimile (214) 593-9111

* Admitted pro hac vice

Counsel for Plaintiff
CYWEE GROUP LTD.

CERTIFICATE OF SERVICE

The undersigned certifies that, on December 29, 2017, a true and correct copy of the foregoing document was served, via email, upon the following counsel of record for Defendants:

Gregory Watts (gwatts@wsgr.com)
WILSON SONSINI GOODRICH & ROSATI
701 Fifth Avenue, Suite 5100
Seattle, WA 98104-7036

James C. Yoon (jyoon@wsgr.com)
Jamie Y. Otto (jotto@wsgr.com)
Albert Shih (ashih@wsgr.com)
Ryan Smith (rsmith@wsgr.com)
WILSON SONSINI GOODRICH & ROSATI
650 Page Mill Road
Palo Alto, CA 94304

/s/ William D. Ellerman
William D. Ellerman

EXHIBIT 1

CYWEE GROUP LTD,
vs.
HTC CORPORATION; AND
HTC AMERICA, INC.

UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WASHINGTON
AT SEATTLE

EXEMPLARY CLAIM CHART

U.S. PATENT NO. 8,441,438 – HTC 10
Infringement Contentions

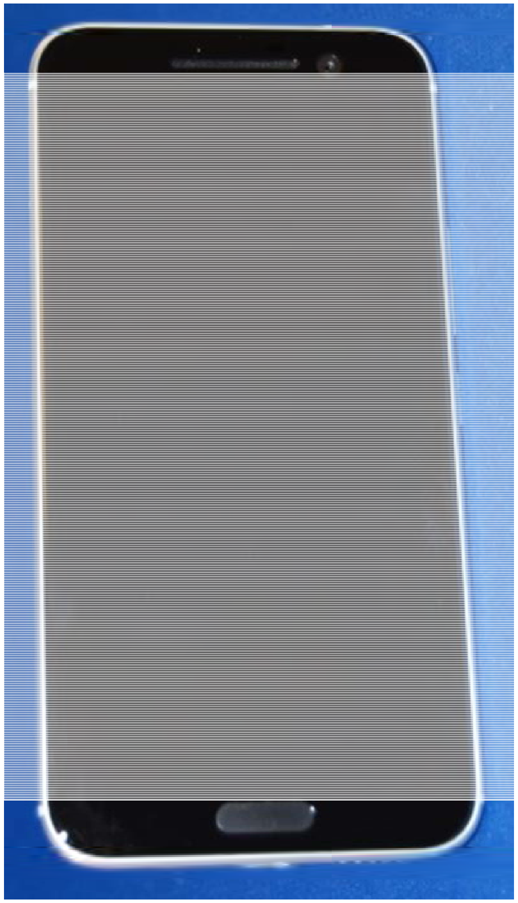
These contentions are disclosed to only provide notice of Plaintiff's theories of infringement. These contentions do not constitute proof nor do they marshal Plaintiff's evidence of infringement to be presented during trial.

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
Claim 1, with claim constructions, is recited below (text in brackets [] reflects the Court's claim construction or the parties' agreed claim construction in *CyWee Group, Ltd. v. Apple Inc.*, No. 3:13-cv-01853-HSG). Construed terms and constructions are underlined. 1. A three-dimensional (3D) pointing device subject to movements and rotations in dynamic environments, comprising: a housing associated with said movements and rotations of the 3D pointing device in a spatial pointer reference frame; a printed circuit board (PCB) enclosed by the housing; a six-axis motion sensor module attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame, an accelerometer for detecting and generating a second signal set comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and a processing and transmitting module, comprising a data transmitting unit electrically connected to the six-axis motion sensor module for transmitting said first and second signal sets thereof and a computing processor for <u>receiving and calculating said first and second signal sets from the data transmitting unit [Court's construction: no construction necessary]</u>, communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by <u>utilizing a comparison to compare the first signal set with the second signal set [Court's construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set]</u> whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments, wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an updated state based on a previous state associated with said first signal set and a measured state associated with said second signal set; wherein the measured state includes a measurement of said second signal set and a predicted measurement obtained based on the first signal set without using any derivatives of the first signal set.

U.S. Patent No. 8,441,438 – HTC 10

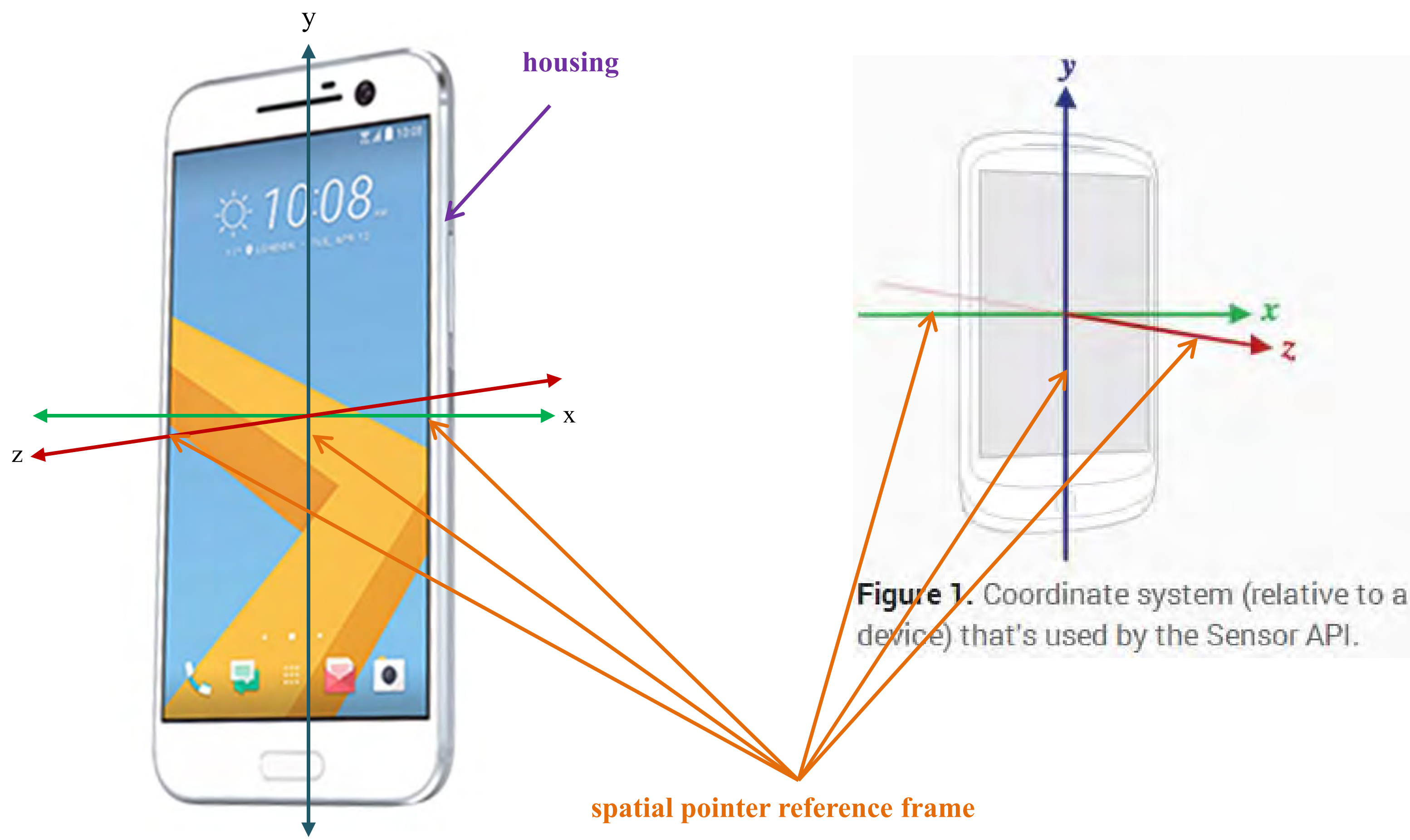
Claim 1
A three-dimensional (3D) pointing device subject to movements and rotations in dynamic environments, comprising:



HTC 10

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a **housing** associated with said movements and rotations of the 3D pointing device in a **spatial pointer reference frame**;

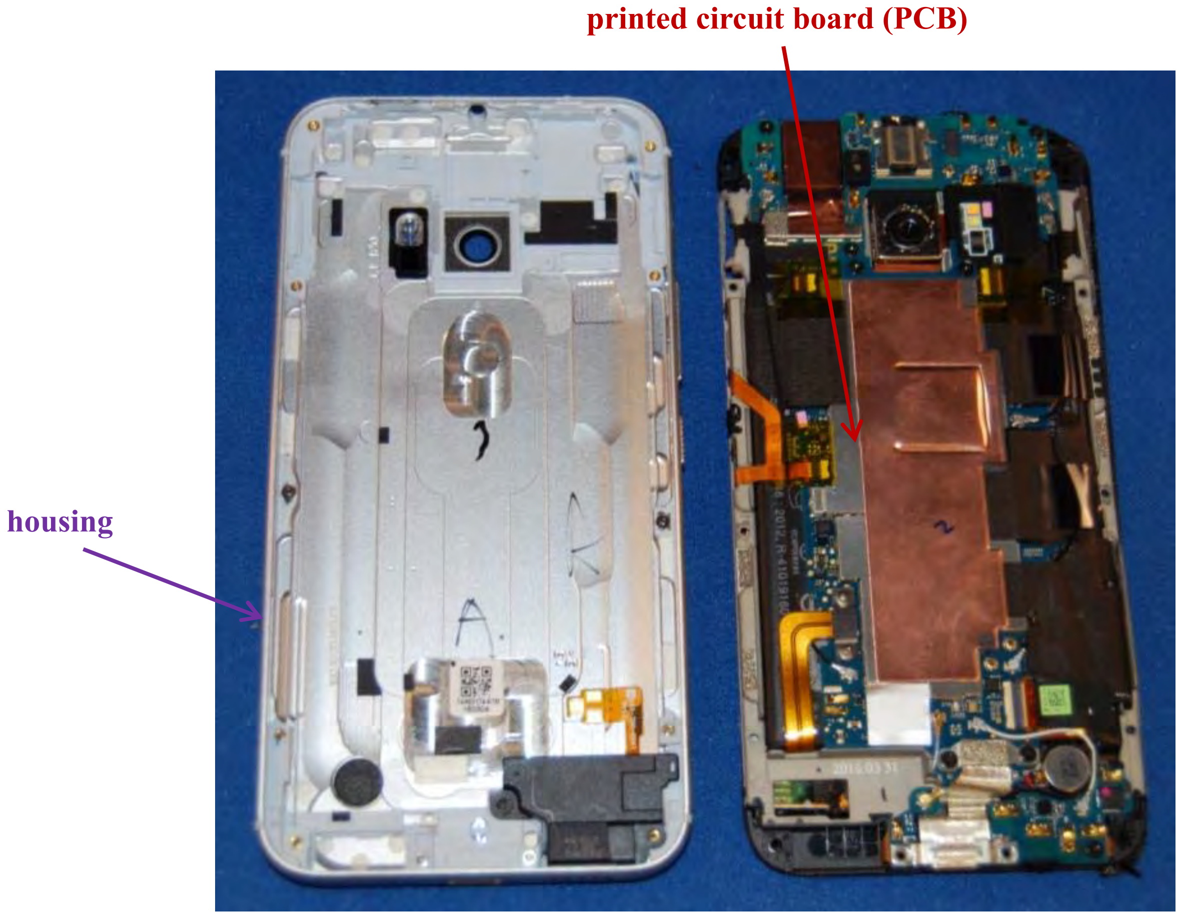


Source: http://www.htc.com/managed-assets/shared/desktop/smartphones/htc-10/explorer/htc-10-global-glacier-silver-angled-listing.png

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

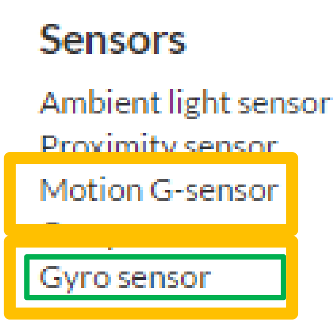
U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a **printed circuit board (PCB)** enclosed by the **housing**;



U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,
The **six-axis motion sensor module** is an accelerometer and gyroscope combo. The **rotation sensor** is the "Gyro sensor" (gyroscope) included in the **six-axis motion sensor module**.



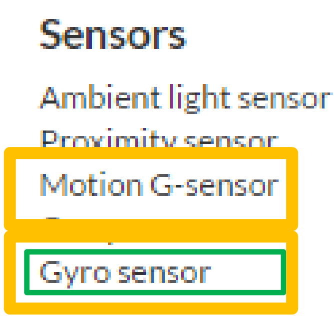
Source: http://www.htc.com/us/smartphones/htc-10/

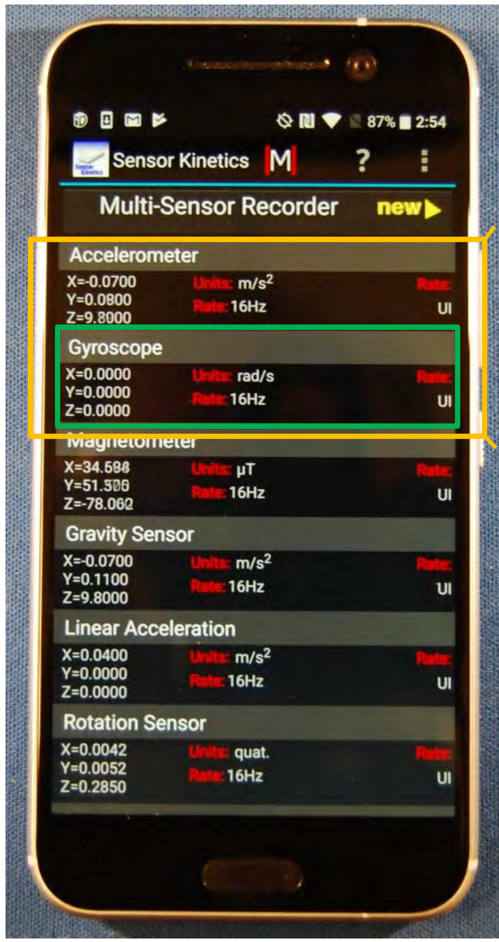
U.S. Patent No. 8,441,438 – HTC 10

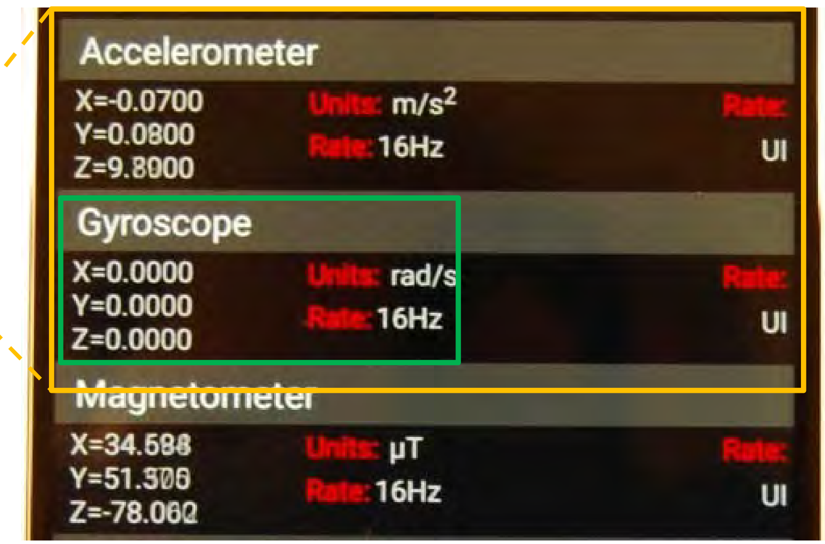
Claim 1

a six-axis motion sensor module attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

The six-axis motion sensor module is an accelerometer and gyroscope combo. The rotation sensor is the "Gyro sensor" (gyroscope) included in the six-axis motion sensor module.







Source: http://www.htc.com/us/smartphones/htc-10/

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**,
The **six-axis motion sensor module** includes the accelerometer and gyroscope. The **rotation sensor** is a gyroscope. The **first signal set** includes the sensor event values of TYPE_GYROSCOPE.

Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_GYROSCOPE)` *returns a non-wake-up sensor*

A gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes.

Rotation is positive in the counterclockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the aerospace definition of roll.

The measurement is reported in the x, y and z fields of `sensors_event_t.gyro` and all values are in radians per second (rad/s).

Source: https://source.android.com/devices/sensors/sensor-types#gyroscope

`Sensor.TYPE_GYROSCOPE`:

All values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis. The coordinate system is the same as is used for the acceleration sensor. Rotation is positive in the counter-clockwise direction. That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the definition of roll given earlier.

- values[0]: Angular speed around the x-axis
- values[1]: Angular speed around the y-axis
- values[2]: Angular speed around the z-axis

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a six-axis motion sensor module attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

Variable `w`, used by the `handleGyro()` function in the fusion.cpp file, represents gyroscope data or a **first signal set**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 1

a six-axis motion sensor module attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**,

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

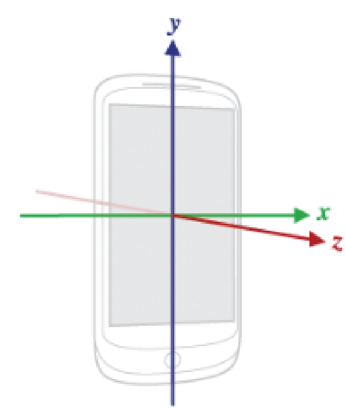


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

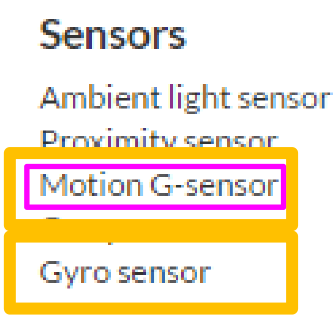
The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

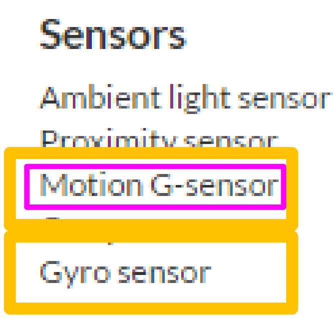
Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and
The **six-axis motion sensor module** is an **accelerometer** and gyroscope combo.

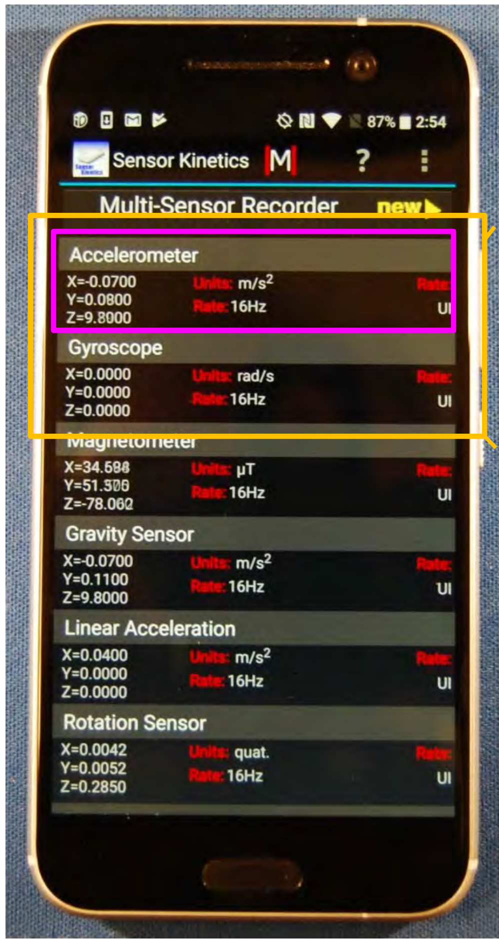


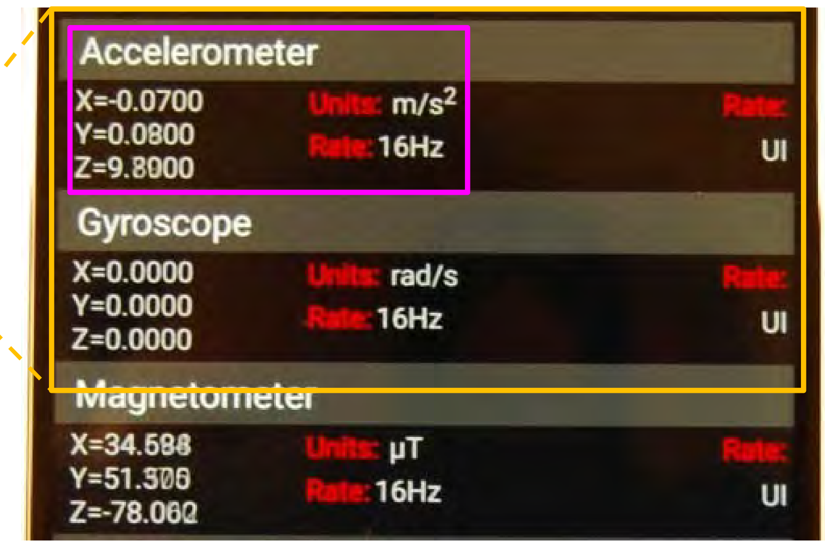
Source: http://www.htc.com/us/smartphones/htc-10/

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and
The **six-axis motion sensor module** is an **accelerometer** and gyroscope combo.







Source: http://www.htc.com/us/smartphones/htc-10/

U.S. Patent No. 8,441,438 – HTC 10

Claim 1

a six-axis motion sensor module attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

The six-axis motion sensor module also includes an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations. The **second signal set** includes the sensor event values of TYPE_ACCELEROMETER.

Accelerometer

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ACCELEROMETER)` *returns a non-wake-up sensor*

An accelerometer sensor reports the acceleration of the device along the 3 sensor axes. The measured acceleration includes both the physical acceleration (change of velocity) and the gravity. The measurement is reported in the x, y and z fields of sensors_event_t.acceleration.

All values are in SI units (m/s^2) and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Source: https://source.android.com/devices/sensors/sensor-types#accelerometer

`Sensor.TYPE_ACCELEROMETER`:

All values are in SI units (m/s^2)

- values[0]: Acceleration minus Gx on the x-axis
- values[1]: Acceleration minus Gy on the y-axis
- values[2]: Acceleration minus Gz on the z-axis

A sensor of this type measures the acceleration applied to the device (**Ad**). Conceptually, it does so by measuring forces applied to the sensor itself (**Fs**) using the relation:

$$\mathbf{Ad = - \sum Fs / mass}$$

In particular, the force of gravity is always influencing the measured acceleration:

$$\mathbf{Ad = -g - \sum F / mass}$$

For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of **g** = 9.81 m/s^2

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 1
a six-axis motion sensor module attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

Variable `a`, used by the `handleAcc()` function in the fusion.cpp file, represents acceleration data or a **second signal set**.

```
status_t Fusion::handleAcc(const vec3_t& a, float dT) {
    if (!checkInitComplete(ACC, a, dT))
        return BAD_VALUE;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a six-axis motion sensor module attached to the PCB, comprising…an accelerometer for detecting and generating a second signal set comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**; and

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

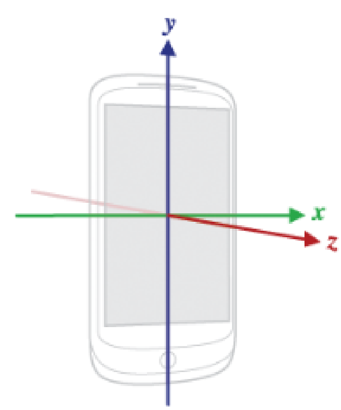


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

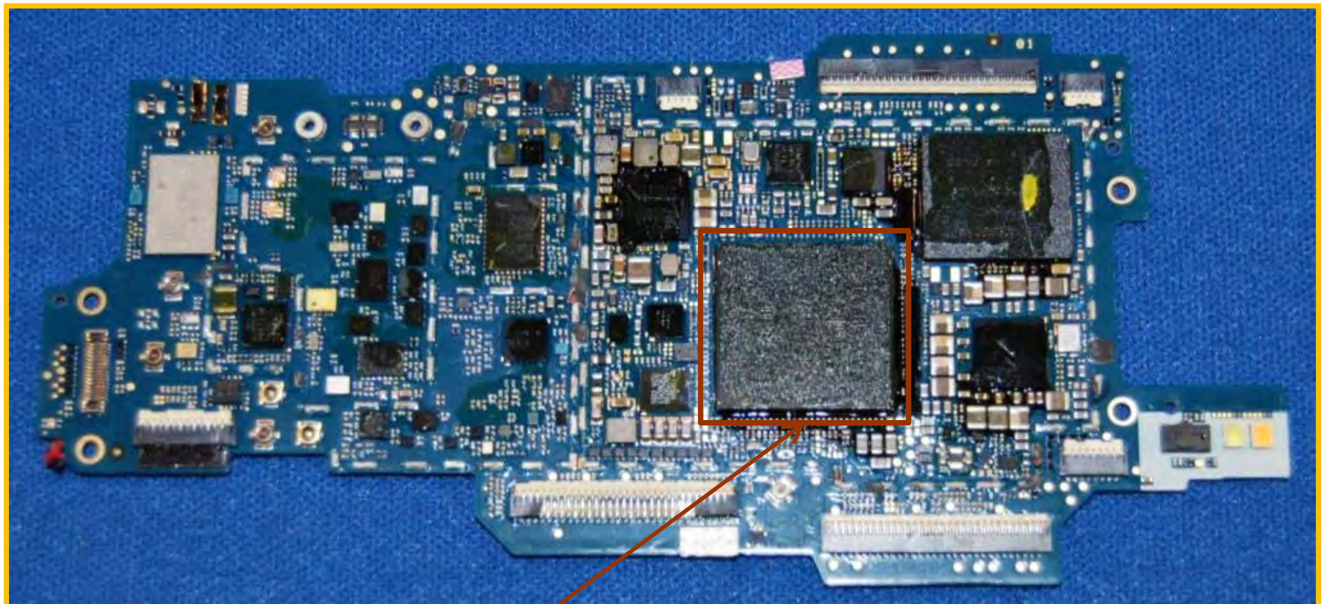
Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
a processing and transmitting module, comprising a **data transmitting unit** electrically connected to the **six-axis motion sensor module** for transmitting said first and second signal sets thereof and a **computing processor** for receiving and calculating said first and second signal sets from the data transmitting unit,

The **computing processor** (Snapdragon 820) gathers data from the **six-axis motion sensor module** (including the accelerometer and the gyroscope) through a **data transmitting unit** (Snapdragon Sensor Core) which is electrically connected to the six-axis motion sensor module.



computing processor

CPU Speed

Qualcomm® Snapdragon™ 820, Quad Core, 64bit, up to 2.2GHz

six-axis motion sensor module

Sensors

Ambient light sensor
Proximity sensor
Motion G-sensor
Compass sensor
Gyro sensor
Magnetic sensor
Fingerprint sensor
Sensor Hub

data transmitting unit

Source: http://www.htc.com/us/smartphones/htc-10/

U.S. Patent No. 8,441,438 – HTC 10

Claim 1

communicating with the six-axis motion sensor module to calculate a **resulting deviation comprising resultant angles** in said spatial pointer reference frame by utilizing a comparison to compare the first signal set with the second signal set [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

getRotationMatrixFromVector

added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrixFromVector(float[], float[])

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Claim 1
communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by <u>utilizing a comparison to compare the **first signal set** with the **second signal set** [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set]</u> whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

The `predict()` function shows that the **first signal set** (angular velocities), `w`, is used to calculate the global variable `x0`.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
    x0 = O*q;
```

The **second signal set** (axial accelerations) `a`, is passed to the variable `z`, and used in the `update()` function to update the global variable `x0`.

```
    vec3_t unityA = a * l_inv;
    update(unityA, Ba, p);
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
    const vec3_t e(z - Bb);
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by utilizing a **comparison to compare** the **first signal set** with the **second signal set** [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

The `predict()` function and `update()` functions are used in sensor fusion to update the global variable `x0` in a quaternion form, which can represent actual deviation angles. In the `predict()` function, the **first signal set**, `w`, is used to calculate the global variable `x0`. In the `update()` function, `x0` is converted to the variable `Bb`. The **second signal set**, `a`, is passed to the `update()` function as local variable z, and is used by the `update()` function to update the global variable `x0`. The variable `Bb` (from the **first signal set**) and the variable `z` (from the **second signal set**) are **compared** to calculate the variable `e` on line 529 of the Fusion.cpp file. Therefore, during the calculation of actual deviation angles, the first signal set is compared with the second signal set.
```430 void Fusion::predict(const vec3_t& w, float dT) {```
```431   const vec4_t q = x0;```
```485   x0 = O*q;```
```495 void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {```
```496   vec4_t q(x0);```
```497   // measured vector in body space: h(p) = A(p)*Bi```
```498   const mat33_t A(quatToMatrix(q));```
```499   const vec3_t Bb(A*Bi);```
```529   const vec3_t e(z - Bb);```
```533   x0 = normalize_quat(q);```
 	
Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an **updated state** based on a **previous state** associated with said **first signal set** and a **measured state** associated with said **second signal set**;
For example, the update program includes a `predict()` function and an `update()` function that are used to update the global variable `x0` based on `x0` (the **previous state**) associated with the **first signal set** `w` and `e` (the **measured state**) associated with the **second signal set** to calculate an **updated state** `x0`. The updated state `x0` becomes the previous state `x0` in the next iteration of the update program to obtain the updated state `x0` in that iteration.

first signal set

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
     x0 = O*q;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

measured state

second signal set

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

next iteration

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 1
wherein the **measured state** includes a **measurement of said second signal set** and a **predicted measurement** obtained based on the first signal set **without using any derivatives of the first signal set**.

The variable e is a **measured state** that includes a **measurement of said second signal set** z and a **predicted measurement** Bb calculated based on x0 (the previous state, which is calculated based on the first signal set).

second signal set (measured accelerations)

```
    vec3_t unityA = a * l_inv;

void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);

    const vec3_t e(z - Bb);
```

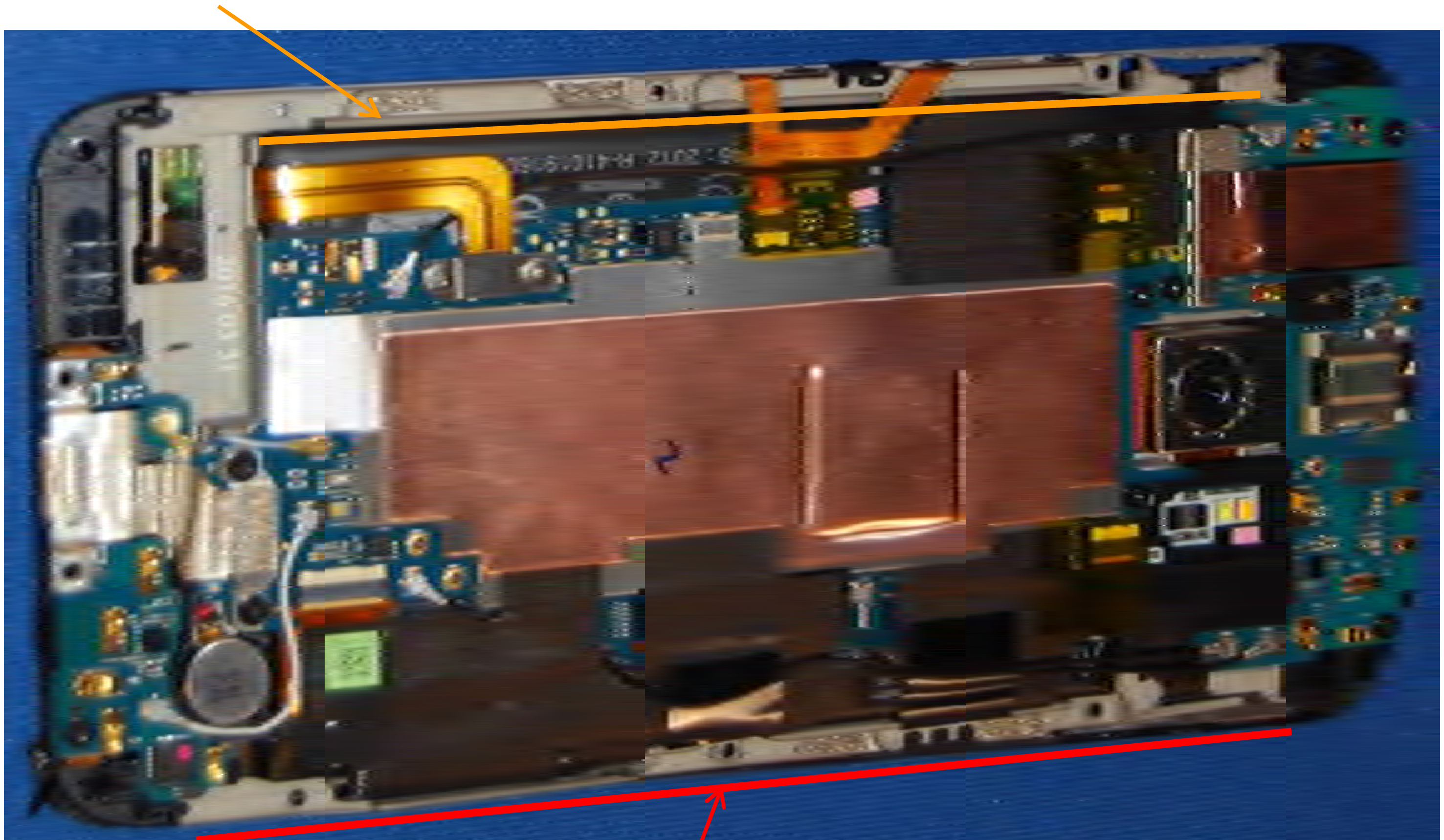
measured state **second signal set** **predicted measurement**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the first signal set**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 3
The 3D pointing device of claim 1, wherein the PCB enclosed by the housing comprises at least one substrate having a **first longitudinal side** configured to be substantially parallel to a **longitudinal surface of the housing**.



U.S. Patent No. 8,441,438 – HTC 10

Claim 4

The 3D pointing device of claim 1, wherein the spatial pointer reference frame is a reference frame in three dimensions; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth, angle of rotation about the -z axis.* This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch, angle of rotation about the x axis.* This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll, angle of rotation about the y axis.* This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

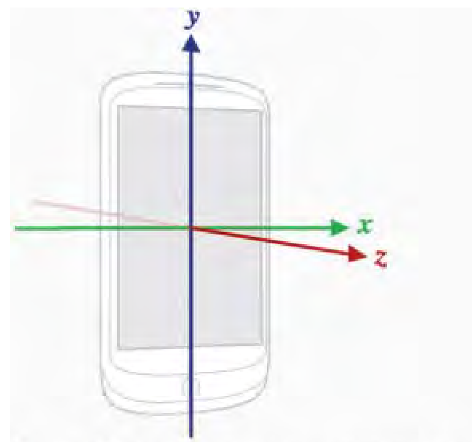
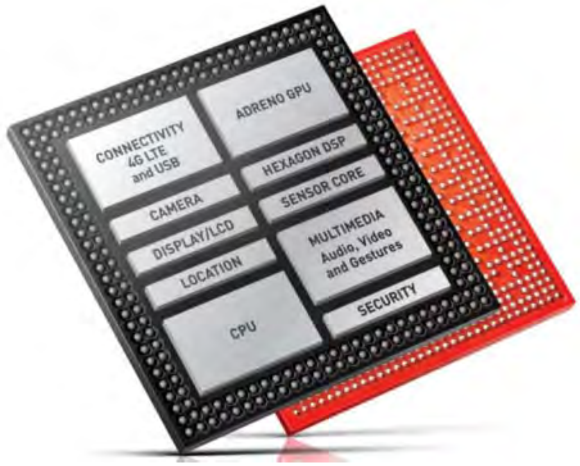


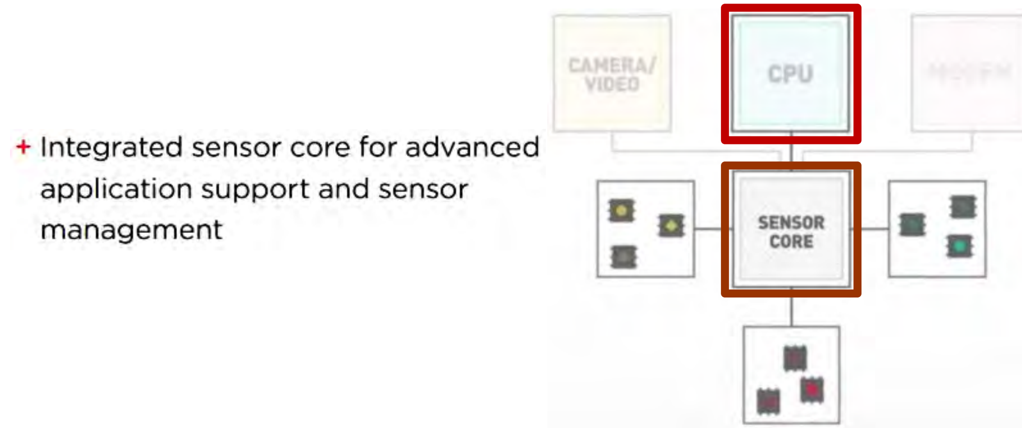
Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

Claim 5
The 3D pointing device of claim 1, wherein the **data transmitting unit** of the processing and transmitting module is attached to the PCB enclosed by the housing and transmits said first and second signal of the six-axis motion sensor module to the **computer processor** via electronic connections.





Source: https://www.qualcomm.com/documents/snapdragon-810-processor-product-brief

Source: https://www.qualcomm.com/videos/get-more-integrated-sensor-engine

U.S. Patent No. 8,441,438 – HTC 10

Claim 14
A method for obtaining a resulting deviation including resultant angles in a **spatial pointer reference frame** of a **three-dimensional (3D) pointing device** utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said **spatial pointer reference frame**, comprising the steps of:

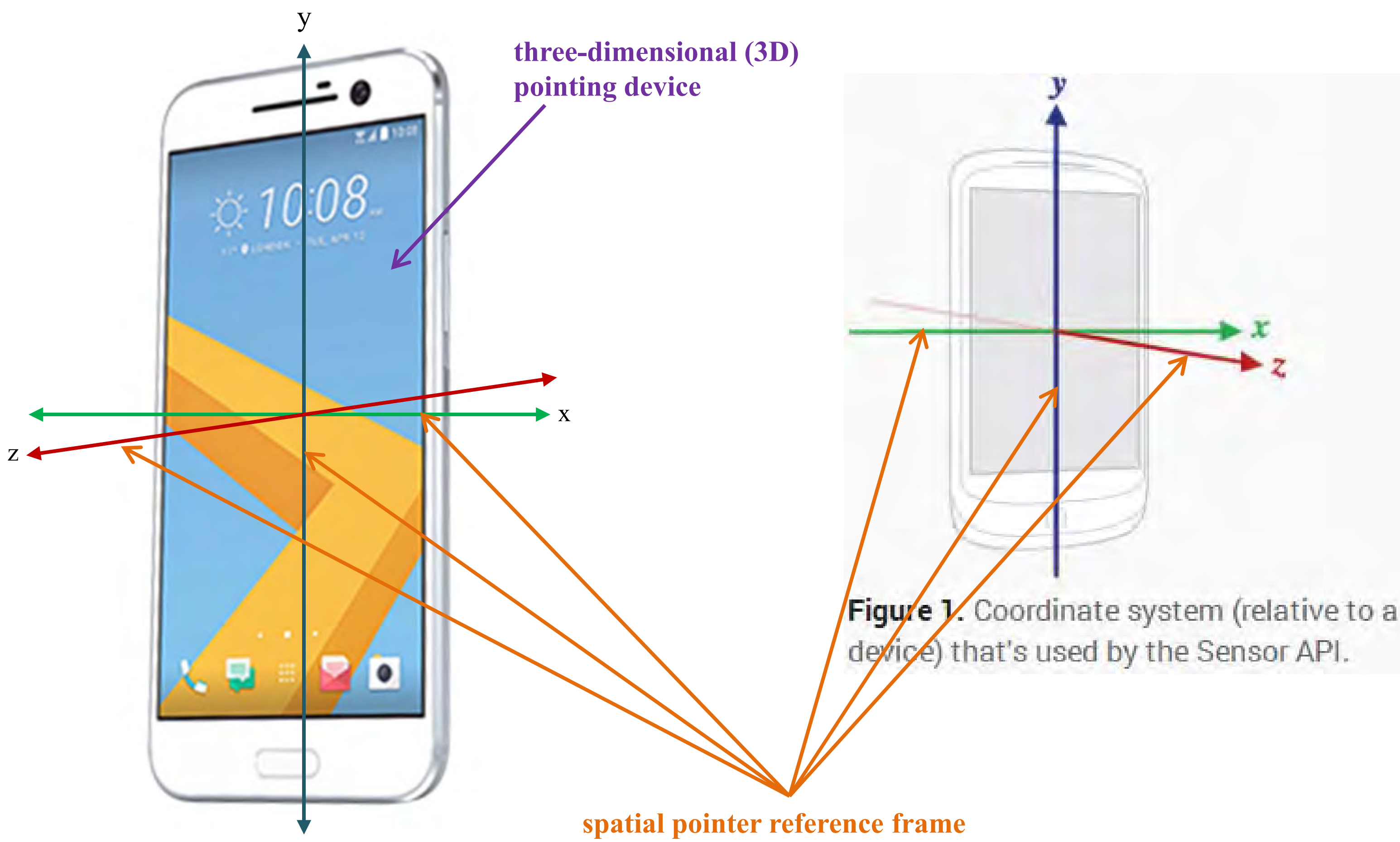


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://www.htc.com/managed-assets/shared/desktop/smartphones/htc-10/explorer/htc-10-global-glacier-silver-angled-listing.png

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

Claim 14

obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable `x0` based on `x0` (the **previous state**) associated with **previous angular velocities** `w` gained at a previous time T-1 to obtain an updated state `x0`. The updated state `x0` becomes the previous state `x0` at time T (the next iteration) of the update program to obtain the updated state `x0` at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
     x0 = O*q;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);

        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 14
obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;
The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as `x0`. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, `w`, associated with the **current state**. **measured angular velocities** **current state** **Source**: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;

void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
     x0 = O*q;
```

U.S. Patent No. 8,441,438 – HTC 10

Claim 14

obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
    vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
```

measured state **measured axial accelerations** **predicted axial accelerations**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 14
said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;
As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 14

comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and

For example, as previously shown, the **measured state**, `e`, is obtained using the `update()` function, which combines the **measured axial accelerations**, `z`, and the **predicted axial accelerations**, `Bb`. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, `e`, and the **current state** to obtain the **updated state**, `x0`.

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
    x0 = O*q;
```

current state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

measured state

measured axial accelerations

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

predicted axial accelerations

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 14
calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.

The **updated state** x0 is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, x0, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Claim 15
The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the **updated state** of the six-axis motion sensor module to the **previous state** of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

For example, Android's source code discloses an iterative process for updating device motion. The **updated state** `x0` output at time T-1 becomes an input of the **previous state** at time T and the "state" is iteratively updated.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

updated state

```
x0 = normalize_quat(q);
```

next iteration

Moreover, the `getOrientation()` function outputs **yaw, pitch and roll angles**.

```
    public static float[] getOrientation(float[] R, float values[]) {
        if (R.length == 9) {
            values[0] = (float)Math.atan2(R[1], R[4]);
            values[1] = (float)Math.asin(-R[7]);
            values[2] = (float)Math.atan2(-R[6], R[8]);
        } else {
            values[0] = (float)Math.atan2(R[1], R[5]);
            values[1] = (float)Math.asin(-R[9]);
            values[2] = (float)Math.atan2(-R[8], R[10]);
        }
```

Source: https://android.googlesource.com/platform/frameworks/base/+/b267554/core/java/android/hardware/SensorManager.java

U.S. Patent No. 8,441,438 – HTC 10

Claim 15

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the updated state of the six-axis motion sensor module to the previous state of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth, angle of rotation about the -z axis.* This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch, angle of rotation about the x axis.* This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll, angle of rotation about the y axis.* This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

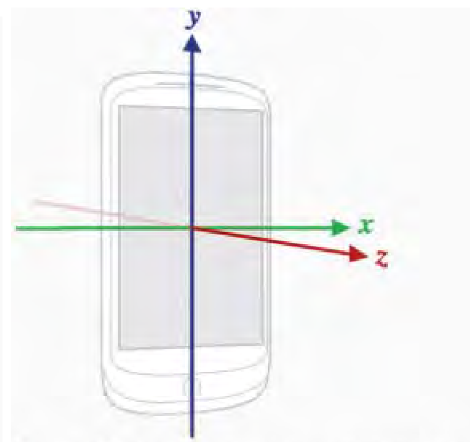


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

Claim 16
The method for obtaining a resulting deviation of a 3D pointing device of claim 14, wherein said **previous state** of the six-axis motion sensor module is a **first quaternion** with respect to said previous time T−1; and said **updated state** of the six-axis motion sensor module is a **third quaternion** with respect to said current time T.
The **previous state** set by the `predict()` function takes the form of a **first quaternion**, `x0`.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

The `update()` function calculates a **third quaternion** representing the **updated state**, `x0`.

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

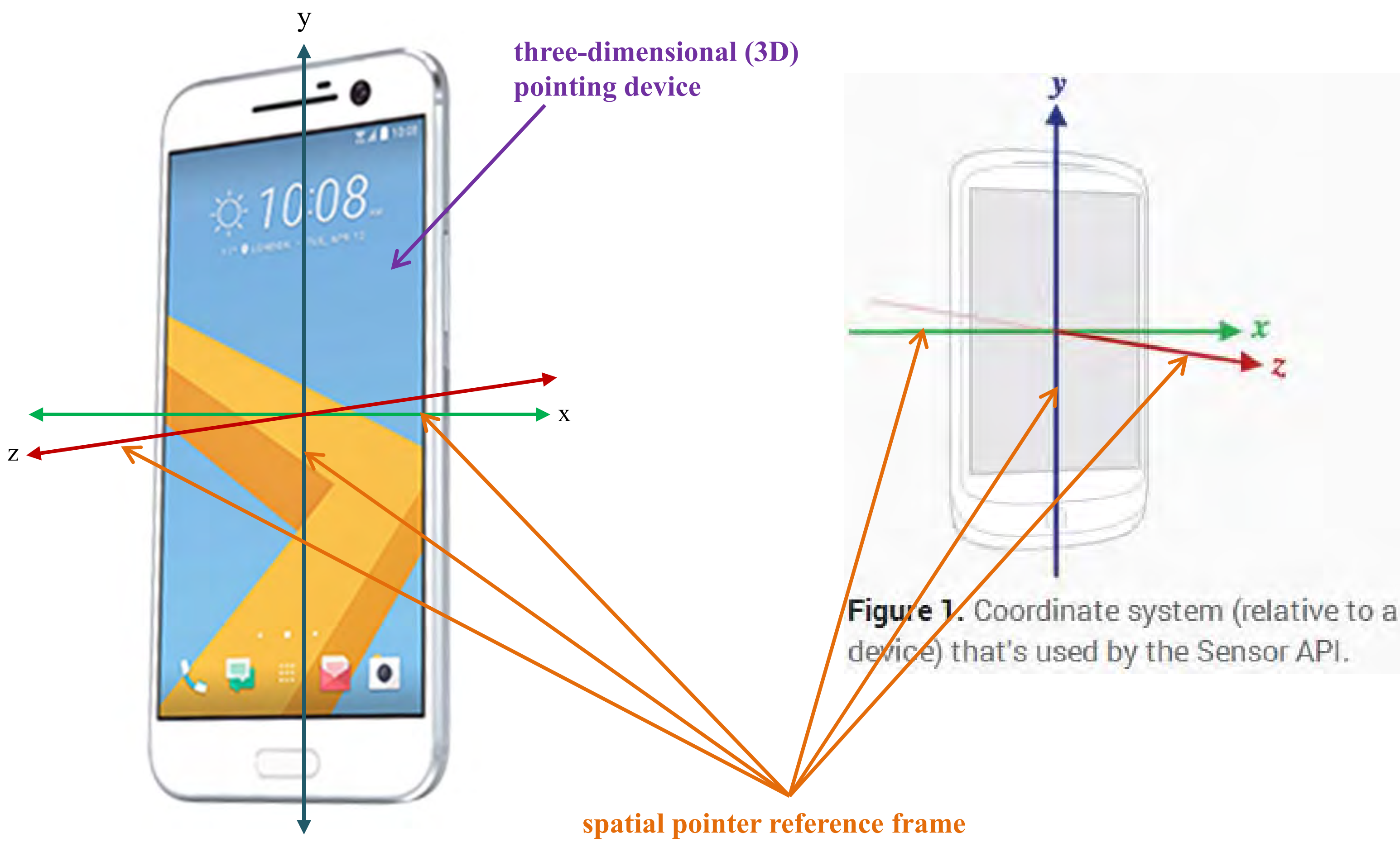
Claim 17
The method for obtaining a resulting deviation of 3D pointing device of claim 14, wherein the obtaining of said previous state of the six-axis motion sensor module further comprises initializing said **initial-value set**.
The fusion algorithm sets an **initial-value set** as shown in the `initFusion()` function.

```
void Fusion::initFusion(const vec4_t& q, float dT)
{
    // initial estimate: E{ x(t0) }
    x0 = q;
    x1 = 0;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19
A method for obtaining a resulting deviation including resultant angles in a **spatial pointer reference frame** of a **three-dimensional (3D) pointing device** utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said **spatial pointer reference frame**, comprising the steps of:



Source: http://www.htc.com/managed-assets/shared/desktop/smartphones/htc-10/explorer/htc-10-global-glacier-silver-angled-listing.png

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

U.S. Patent No. 8,441,438 – HTC 10

Claim 19

obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable `x0` based on `x0` (the **previous state**) associated with **previous angular velocities** `w` gained at a previous time T-1 to obtain an updated state `x0`. The updated state `x0` becomes the previous state `x0` at time T (the next iteration) of the update program to obtain the updated state `x0` at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
     x0 = O*q;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19
obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;
The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as `x0`. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, `w`, associated with the **current state**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;

void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
     x0 = O*q;
```

measured angular velocities

current state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19
obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
    vec3_t unityA = a * l_inv;

void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);

    const vec3_t e(z - Bb);
```

measured state **measured axial accelerations** **predicted axial accelerations**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19

said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;

As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19
comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and

For example, as previously shown, the **measured state**, `e`, is obtained using the `update()` function, which combines the **measured axial accelerations**, `z`, and the **predicted axial accelerations**, `Bb`. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, `e`, and the **current state** to obtain the **updated state**, `x0`.
 	
measured angular velocities
`void Fusion::predict(const vec3_t& w, float dT) {`
`    const vec4_t q  = x0;` ← **previous state**
`    x0 = O*q;` ← **current state**
 	
`void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {`
`    vec4_t q(x0);`
 	
measured state
measured axial accelerations
`    const vec3_t e(z - Bb);`
`    const vec3_t dq(K[0]*e);`
 	
predicted axial accelerations
`    q += getF(q)*(0.5f*dq);`
updated state → `    x0 = normalize_quat(q);`
 	
Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

U.S. Patent No. 8,441,438 – HTC 10

Claim 19
calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.

The **updated state** x0 is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the getOrientation() function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The getRotationMatrixFromVector() function "convert[s] a rotation vector to a rotation matrix," and the getQuaternionFromVector() function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, x0, can be easily converted to its mathematically equivalent form, rotation matrix, and used by getOrientation() function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])